Gestione di barre strumenti e riga di stato

La volta scorsa abbiamo presentato la classe TNewMDIFrame, *illustrando variabili d'istanza e metodi che presiedono alla creazione della* frame window *ed alla determinazione del suo assetto, alla gestione delle* child window, *alla visualizzazione nella riga di stato di messaggi, compresi quelli esplicativi delle opzioni dei menu. Rimane da esaminare la gestione di barre strumenti e riga di stato, apportando peraltro qualche modifica alla unit TOOLBAR, per rendere possibile la visualizzazione di messaggi esplicativi sulla riga di stato quando l'utente si posiziona su un pulsante della barra strumenti*

di Sergio Polini

In occasione dell'appuntamento di maggio, avevamo rilevato che la unit TOOLBAR fornita insieme al Borland Pascal 7.0 non consente di dotare un'applicazione di più di una riga di stato e non rende possibile nascondere l'unica barra consentita.

Il mese successivo, quindi, avevamo modificato quella unit aggiungendo le costanti *tbHidden* e *tbDownHorizontal*, riscrivendo completamente il metodo *AMCalcParentClientRect*, apportando qualche ritocco al constructor *Init* ed ai metodi *NextToolOrigin*, *ReadResource*, *SetOrientation* e *Paint*.

Nei mesi scorsi abbiamo illustrato alcune unit utili per realizzare applicazioni MDI dotate di barre strumenti e riga di stato, nelle quali la riga di stato possa essere utilizzata, tra l'altro, per mostrare messaggi esplicativi delle opzioni dei menu.

Il meccanismo risulta piuttosto flessibile e si presta anche alla visualizzazione di messaggi che illustrino il ruolo dei pulsanti di una barra strumenti; a questo scopo, però, è necessario apportare qualche ulteriore ritocco alla unit TOOLBAR.

Barre parlanti

Nostro obiettivo è disporre di barre strumenti che ci comunichino, attraverso la riga di stato, il significato dei loro pulsanti. Occorre, quindi, che ad essi non sia associato unicamente un comando, ma anche un codice di «aiuto», analogo a quelli che i metodi *WMMenuSelect* delle classi *TNewMDIChild* e *TNewMDIFrame* associano alle opzioni dei menu. Ricordiamo che una barra strumenti viene definita nel file di risorse mediante una risorsa di tipo TOOLBARDATA, che non appartiene al corredo standard e, quindi, può essere creata solo mediante un file di testo con estensione RC. La descrizione della barra inizia con un numero che indica il numero dei componenti, i quali possono essere di due tipi, entrambi espressi mediante coppie di numeri. Se si tratta di un normale pulsante, il primo numero denota la bitmap del pulsante ed il secondo il comando ad esso associato; se il primo numero è zero non c'è bitmap, ma uno spazio tra due pulsanti, la cui ampiezza è indicata dal secondo numero. Per aggiungere una costante di «aiuto», occorre modificare in terne le coppie che definiscono i pulsanti e gli spazi tra pulsanti.

Il *Resource Workshop* consente di scrivere i file RC includendo file contenenti la dichiarazione di costanti (attraverso l'opzione *Add to project* del menu *File*). Ciò consente di proporre un esempio utilizzando costanti simboliche dichiarate nei file OWINDOWS.INC e MDICONST.PAS.

Supponiamo, quindi, di volere una barra strumenti FILETOOLBAR con pulsanti corrispondenti ai comandi *cm_MDIFileOpen*, *cm_FileSave* e *cm_Print* (quest'ultimo separato dai due precedenti), utilizzando bitmap identificate dai numeri 1000, 1001 e 1002. Dovremo creare un file RC contenente la dichiarazione riprodotta nella figura 1; in essa ogni pulsante viene definito mediante tre numeri, identificativi della bitmap, del comando e della costante di aiuto; quanto a quest'ultima, si usa lo stesso valore che verrebbe calcolato da un metodo *WMMenuSelect* (in pratica, come mostra l'esempio, basta aggiungere *ids_MenuItem* al comando).

La nuova struttura della risorsa TOOLBARDATA impone di modificare, innanzitutto, il constructor della classe *TToolButton* (figura 2) e i metodi della classe *TToolbar* che leggono la barra dal file di risorse (figura 3).

Aggiungiamo, quindi, una variabile d'istanza *HelpCode* alla classe *TToolButton* e un parametro *AHelpCode* al suo constructor, in modo che questo possa assegnare alla variabile il valore passato attraverso il parametro.

Quanto a *TToolbar*, teniamo conto di tali modifiche aggiungendo un parametro *HelpCode* al metodo *CreateTool*, affinché questo possa servirsene per chiamare in modo corretto il constructor dei pulsanti. Dobbiamo intervenire

```
#include "owindows.inc"
#include "mdiconst.pas"

FILETOOLBAR TOOLBARDATA
BEGIN
4                                   /* numero componenti    */
1000                                /* File/Nuovo:  bitmap  */
cm_MDIFileOpen                      /*              comando */
ids_MenuItem + cm_MDIFileOpen       /*              aiuto   */
1001                                /* File/Salva:  bitmap  */
cm_FileSave                         /*              comando */
ids_MenuItem + cm_FileSave          /*              aiuto   */
0                                       /*          */
8                                       /* spazio   */
0                                       /*          */
1002                                /* File/Stampa: bitmap  */
cm_Print                            /*              comando */
ids_MenuItem + cm_Print             /*              aiuto   */
END
```

Figura 1 - Struttura della dichiarazione di una barra strumenti con costanti di aiuto associate ai pulsanti.

anche sul metodo *ReadResource*, quello che legge il file di risorse e chiama *CreateTool* con i valori da questo tratti: modifichiamo il record *ResRec* aggiungendo un campo *HelpCode* di tipo *word* e, per ogni pulsante di cui si sia letta la descrizione, chiamiamo *CreateTool* con il nuovo set di parametri.

Il meccanismo di comunicazione

I comandi associati ad un pulsante della barra strumenti vengono eseguiti quando l'utente, dopo essersi posizionato sopra di esso con il mouse, preme il pulsante sinistro e poi lo rilascia, rimanendo sullo stesso pulsante della barra. Il comando non viene eseguito se l'utente rilascia il pulsante sinistro dopo essersi spostato altrove con il mouse.

Questo consente di organizzare la visualizzazione di messaggi esplicativi prevedendone l'apparizione quando l'utente preme il pulsante sinistro e la scomparsa quando lo rilascia, mantenendoli visibili anche se l'utente si sposta con il mouse tenendo premuto il pulsante sinistro (in questo caso, il rilascio del pulsante non provocherà l'esecuzione del comando; la sistemazione ora descritta, quindi, risulta particolarmente utile quando l'utente vuole semplicemente esplorare il funzionamento della barra strumenti, senza attivare alcun comando). La classe *TToolbar* riconosce i movimenti del mouse mediante i metodi *WMLButtonDown, WMMouseMove e WMLButtonUp:* il primo individua il pulsante su cui l'utente ha clickato (riconosciuto perché è quello che risponde affermativamente al metodo *HitTest*), ne assegna l'indirizzo alla variabile *Capture* e ne chiama il metodo *BeginCapture*; il secondo ed il terzo chiamano, rispettivamente, i metodi *ContinueCapture* e *EndCapture* dello stesso pulsante. Per ottenere la visualizzazione di un messaggio esplicativo del pulsante su cui l'utente ha clickato, quindi, è sufficiente modificare il metodo *BeginCapture* della classe *TToolButton* in modo che invii alla *frame window* i messaggi *um_SetHelpCode* e *um_PaintStatusLine*, quest'ultimo con un *WParam* pari a *sl_Plain* ed un *LParam* nullo. Per ottenere che il messaggio scompaia e venga ripristinato il precedente contenuto della riga di stato, è sufficiente modificare il metodo *EndCapture*, aggiungendo l'invio alla *frame window* di un messaggio WM_MENUSELECT con *WParam* nullo e *LParam* pari a $0000FFFF, cioè dello stesso messaggio che viene generato da Windows quando l'utente abbandona un menu. Le modifiche sono illustrate nella figura 4.

Visibilità e orientamento

Con le modifiche appena viste, la unit TOOLBAR è pronta per essere utilizzata insieme alle unit dedicate alla realizzazione di applicazioni MDI. Possiamo quindi completare l'illustrazione di queste ultime, esaminando l'ultimo gruppo di metodi della classe *TNewMDIFrame* (figura 5).

Ricordiamo che una barra di stato può essere visibile o nascosta e che, se visibile, può essere collocata lungo uno qualsiasi dei bordi della finestra; sembra ragionevole assumere che, se l'utente na-

```
TToolButton = object(TTool)
  (* ... *)
  HelpCode: Word;
  (* ... *)
  constructor Init(AParent: PWindowsObject; X, Y: Integer;
                   ACommand, AHelpCode: Word; BitmapName: PChar);
  (* ... *)
end;

constructor TToolButton.Init(AParent: PWindowsObject; X,Y:Integer;
                   ACommand, AHelpCode: Word; BitmapName: PChar);
var
  (* ... *)
begin
  (* ... *)
  HelpCode := AHelpCode;
  (* ... *)
end;
```

Figura 2 - Le modifiche da apportare alla classe TToolButton *nella unit TOOLBAR.*

```
TToolbar = object(TWindow)
  (* ... *)
  function  CreateTool(Num: Integer; Origin: TPoint;
                       Command, HelpCode: Word;
                       BitmapName: PChar): PTool; virtual;
  (* ... *)
end;

function TToolbar.CreateTool(Num: Integer; Origin: TPoint;
                             Command, HelpCode: Word;
                             BitmapName: PChar): PTool;
begin
  if Word(BitmapName) = 0 then
    CreateTool := New(PToolSpacer, Init(@Self, Command))
  else
    CreateTool := New(PToolButton, Init(@Self, Origin.X, Origin.Y,
                                        Command, HelpCode,
                                        BitmapName));
end;

procedure TToolbar.ReadResource;
type
  ResRec = record
    Bitmap,
    Command: Word;
    HelpCode: Word;
  end;
  (* ... *)
var
  (* ... *)
begin
  (* ... *)
  for X := 1 to Count do
    with ResDataPtr^[X] do begin
      P := CreateTool(X,Origin,Command,HelpCode,PChar(Bitmap));
      (* ... *)
    end;
  (* ... *)
end;
```

Figura 3 - Le modifiche da apportare ai metodi della classe TToolbar *che leggono i pulsanti dal file di risorse.*

Figura 4 - Le modifiche da apportare ai metodi BeginCapture *e* EndCapture *della classe* TToolButton *per ottenere la visualizzazione di messaggi sulla riga di stato.*

```
procedure TToolButton.BeginCapture(P: TPoint);
begin
  (* ... *)
  SendMessage(Parent^.Parent^.HWindow, um_SetHelpCode,
              HelpCode, 0);
  SendMessage(Parent^.Parent^.HWindow, um_PaintStatusLine,
              sl_Plain, 0);
end;

function TToolButton.EndCapture(SendTo: HWnd; P: TPoint): Boolean;
begin
  if HitTest(P) and IsEnabled then begin
    (* ... *)
  end;
  (* ... *)
  SendMessage(Parent^.Parent^.HWindow, wm_MenuSelect,
              0, $0000FFFF);
end;
```

```
procedure TNewMDIFrame.CMToolbar(var Msg: TMessage);
var
  HM: HMenu;
  Orient: Word;
  MenuFlag: Word;
begin
  if Toolbar = nil then
    Exit;
  HM := GetMenu(HWindow);
  Toolbar^.Show(sw_Hide);
  if Toolbar^.GetOrientation = tbHidden then begin
    Orient := tbHorizontal;
    if (GetMenuState(HM, cm_DownHorizontalToolbar,
                     mf_ByCommand) and mf_Checked) <> 0 then
      Orient := tbDownHorizontal
    else if (GetMenuState(HM, cm_LeftVerticalToolbar,
                     mf_ByCommand) and mf_Checked) <> 0 then
      Orient := tbLeftVertical
    else if (GetMenuState(HM, cm_RightVerticalToolbar,
                     mf_ByCommand) and mf_Checked) <> 0 then
      Orient := tbRightVertical;
    MenuFlag := mf_Checked;
  end
  else begin
    Orient := tbHidden;
    MenuFlag := mf_UnChecked;
  end;
  Toolbar^.SetOrientation(Orient);
  RedoClientRect;
  if Orient <> tbHidden then
    Toolbar^.Show(sw_Show);
  CheckMenuItem(HM, cm_Toolbar, MenuFlag);
end;

procedure TNewMDIFrame.CMStatusLine(var Msg: TMessage);
var
  Orient: Word;
  MenuFlag: Word;
begin
  if StatusLine = nil then
    Exit;
  StatusLine^.Show(sw_Hide);
  if StatusLine^.GetOrientation = tbHidden then begin
    Orient := tbDownHorizontal;
    MenuFlag := mf_Checked;
  end
  else begin
    Orient := tbHidden;
    MenuFlag := mf_Unchecked;
  end;
  StatusLine^.SetOrientation(Orient);
  RedoClientRect;
  if Orient <> tbHidden then
    StatusLine^.Show(sw_Show);
  CheckMenuItem(GetMenu(HWindow), cm_StatusLine, MenuFlag);
end;
```

```
procedure TNewMDIFrame.OrientToolbar(Command: Word);
const
  FlagYes = mf_ByCommand or mf_Checked;
  FlagNo  = mf_ByCommand or mf_Unchecked;
var
  OldOrient, NewOrient: Word;
  Menu: HMenu;
begin
  if Toolbar = nil then
    Exit;
  OldOrient := Toolbar^.GetOrientation;
  case Command of
    cm_HorizontalToolbar    : NewOrient := tbHorizontal;
    cm_DownHorizontalToolbar: NewOrient := tbDownHorizontal;
    cm_LeftVerticalToolbar  : NewOrient := tbLeftVertical;
    cm_RightVerticalToolbar : NewOrient := tbRightVertical;
  end;
  if NewOrient <> OldOrient then begin
    if OldOrient <> tbHidden then begin
      Toolbar^.Show(sw_Hide);
      Toolbar^.SetOrientation(NewOrient);
      RedoClientRect;
      Toolbar^.Show(sw_Show);
    end;
    Menu := GetMenu(HWindow);
    CheckMenuItem(Menu, cm_HorizontalToolbar, FlagNo);
    CheckMenuItem(Menu, cm_DownHorizontalToolbar, FlagNo);
    CheckMenuItem(Menu, cm_LeftVerticalToolbar, FlagNo);
    CheckMenuItem(Menu, cm_RightVerticalToolbar, FlagNo);
    CheckMenuItem(Menu, Command, FlagYes);
  end;
end;

procedure TNewMDIFrame.CMHorizontalToolbar(var Msg: TMessage);
begin
  OrientToolbar(Msg.WParam);
end;

procedure TNewMDIFrame.CMDownHorizontalToolbar(var Msg: TMessage);
begin
  OrientToolbar(Msg.WParam);
end;

procedure TNewMDIFrame.CMLeftVerticalToolbar(var Msg: TMessage);
begin
  OrientToolbar(Msg.WParam);
end;

procedure TNewMDIFrame.CMRightVerticalToolbar(var Msg: TMessage);
begin
  OrientToolbar(Msg.WParam);
end;
```

Figura 5 - L'ultimo gruppo di metodi della classe TNewMDIFrame.

sconde una barra, quando la rende nuovamente visibile desidera vederla nello stesso posto in cui questa era prima di essere nascosta. Ciò comporta che il menu deve proporre sia un'opzione che consenta di visualizzare e nascondere la barra, sia un separato gruppo di opzioni che consenta di determinarne l'orientamento.

Il file MDICONST.PAS, quindi, prevede sia un comando *cm_Toolbar* (che troverebbe la sua sede naturale nel menu *Visualizza*), sia i quattro comandi *cm_HorizontalToolbar, cm_DownHorizontalToolbar, cm_LeftVerticalToolbar* e *cm_RightVerticalToolbar*.

Il metodo *CMToolbar* di *TNewMDIFrame* esegue il comando *cm_Toolbar* rendendo alternativamente visibile e invisibile la barra; nel far ciò, assume che il menu dell'applicazione comprenda anche i comandi di orientamento, anche se, in caso contrario, può comunque operare correttamente.

Se la barra è nascosta (la sua variabile *Orientation* vale *tbHidden*), si assume inizialmente che debba essere visualizzata nella sua posizione «normale» (orizzontale, lungo il bordo superiore della finestra); se, però, risulta che l'utente abbia in precedenza optato per un orientamento diverso - scelta desumibile dallo stato *mf_Checked* della corrispondente opzione del menu - se ne tiene conto, assegnando il valore opportuno alla variabile *Orient* oltre che *mf_Checked* alla variabile *MenuFlag*. Se la barra è invece visibile (la sua variabile *Orientation* ha un valore diverso da *tbHidden*), la si nasconde assegnando *tbHidden* a *Orient* e *mf_UnChecked* a *MenuFlag*.

Si procede poi a modificare l'orientamento della barra passando la variabile Orient al suo metodo *SetOrientation*, si ridisegna la *client area* della *frame window*, si marca l'opzione del menu associata al comando *cm_Toolbar* se l'utente ha chiesto la visualizzazione della barra.

Il metodo *CMStatusLine* opera in modo del tutto analogo, provvedendo a nascondere o visualizzare la riga di stato secondo i desideri dell'utente. Il meccanismo è peraltro più semplice, in quanto per la riga di stato si assume un solo orientamento possibile (orizzontale, lungo il bordo inferiore della finestra).

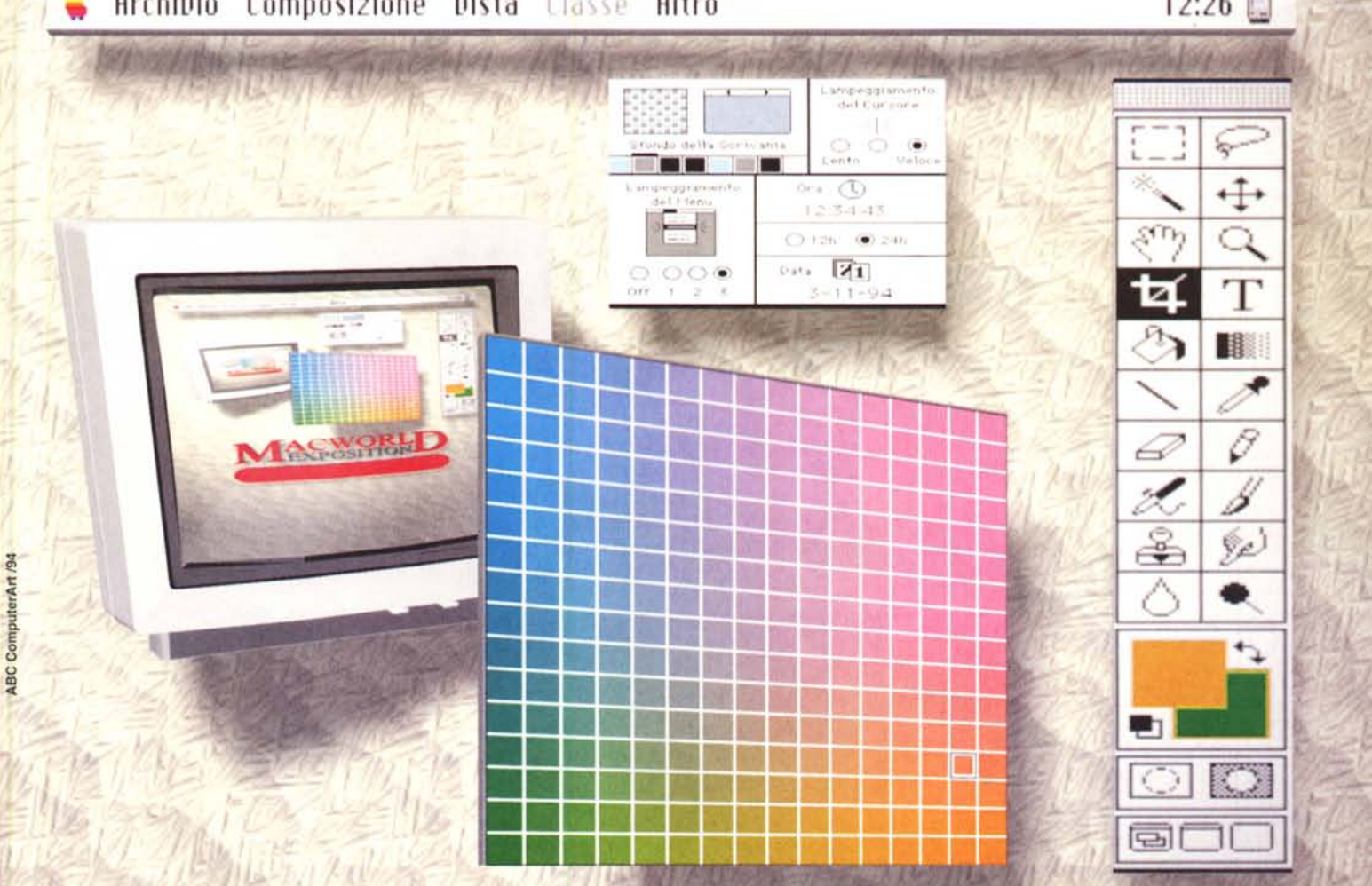
Restano da esaminare i quattro metodi corrispondenti ai comandi mediante i quali l'utente può cambiare l'orientamento della barra strumenti.

Ognuno di essi non fa altro che girare il comando, contenuto nel campo *WParam* del parametro *Msg*, ad un metodo *OrientToolbar*. Questo, dopo aver letto in *OldOrient* l'orientamento attuale della barra, assegna alla variabile *NewOrient* il nuovo orientamento scelto dell'utente; se i due orientamenti sono diversi, e se quello attuale (*OldOrient*) non è *tbHidden*, la barra viene ridisegnata e viene marcata l'opzione del menu corrispondente al nuovo orientamento.

Abbiamo così terminato l'illustrazione di un insieme di unit che semplificano notevolmente la realizzazione di applicazioni MDI.

Per apprezzare in concreto la loro utilità, sarà necessario proporre un «demo», che ragioni di spazio impongono di rimandare al mese prossimo. MC

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo mc1166@mclink.it.



MACWORLD EXPOSITION

Milano, 3-6 Maggio 1995

4ª MOSTRA CONVEGNO DEL MERCATO DEI SISTEMI MACINTOSH®

SPAZIO MILANO*NORD*
Via Pompeo Mariani, 2 - Milano

Orario: 9.30 - 18.00

Macworld Expo '95 è un'iniziativa
ASSOEXPO • IDG

– Macintosh è un marchio registrato di Apple Computer –

Segreteria Generale:
"MACWORLD EXPO"
Via Domenichino, 11
20149 Milano
Tel. 02/4815541
Fax 02/4980330

Segreteria Scientifica:
IDG Communications Italia
Via G. Malipiero, 14
20138 Milano
Tel. 02/58011660
Fax 02/58011670